UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

STUDY OF RECONSTRUCTION ICA FOR FEATURE EXTRACTION IN IMAGES AND SIGNALS

Autor: Marc Beltrán Segarra

Director: Laura Igual Muñoz
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, June 22, 2017

Abstract

During the last years, neural networks have become a vehicular discipline in the field of machine learning. At the same time, classical machine learning methods have become easier to use due to the availability of higher computational power. The goal of this project is to reconstruct a classical machine learning algorithm used for feature and source extraction (ICA) using neural networks. This reconstruction could bypass some of the drawbacks presents when using ICA. We have studied how the reconstruction operates under different conditions and performed a comparison with the classical algorithm that we reconstructed.

Resum

Les xarxes neuronals s'han convertit en una disciplina fonamental a dins del camp de l'aprenentatge autònom o *machine learning*. Al mateix temps, mètodes clàwsics d'aquest camp han passat a ser més fàcils d'utilitzar a causa dels avanços tecnològics. L'objectiu d'aquest projecte és la reconstrucció d'un d'aquests algoritmes clàssics per l'extracció de característiques i fonts (*ICA*) utilitzant xarxes neuronals. Aquesta reconstrucció podria evitar alguns dels inconvenients que existeixen alhora d'utilitzar *ICA*. Hem estudiat com la reconstrucció opera sota diferents condicions i realitzat una comparació amb l'algorisme clàssic que hem reconstruït.

Resumen

Las redes neuronales se han convertido en una disciplina fundamental dentro del campo del aprendizaje automático o *machine learning*. Al mismo tiempo, métodos clásicos de este campo han pasado a ser más fáciles de utilizar debido a los avances tecnológicos. El objetivo de este proyecto es la reconstrucción de uno de estos algoritmos clásicos (*ICA*) usando redes neuronales. Esta reconstrucción podá evitar algunos de los inconvenientes que existen cuando se usa *ICA*. Hemos estudiado como la reconstrucción funciona bajo distintas condiciones y realizado una comparación con el algoritmo clásico que hemos reconstruido.

Contents

1 Introduction

1.1 Context

Historically, tasks such as object or speech recognition have been tough to handle by computers, despite humans being close to perfect with them. This was caused by the lack of powerful hardware which prevented researchers to work in complex programming models that required high computational power.

Nowadays, new types of programming have appeared thanks to technological progress and the widespread availability of powerful hardware such as GPUs. This has allowed programmers and researchers to implement new algorithms and programming methodologies that rely on iterative methods that require a high number of operations done by the computer. These methods can be used to work with high dimensional data, such as images or signals, and tackle problems such as classification or clustering tasks when dealing with image or sound data.

Neural networks are programming models based on emulating how the human brain operates by creating a network of interconnected entities that feed off each other. Despite them being created over 20 years ago, they were not as useful as they are now because the training process had a high computational cost.

On the other hand, classical machine learning techniques such as Principal Component Analysis or Independent Component Analysis have become more available and easier to use too, but still have some limitations.

1.2 Objectives

The objective of this project is to study the relationship between neural network techniques and classical machine learning methods, by studying a reconstruction of Independent Component Analysis (ICA) that is created using structures and properties of neural networks. The study is comprised of a comparison test between the reconstructed version of ICA (RICA) and the original ICA, as well as an analysis on how varying different parameters in RICA affects the results.

1.3 Motivation

The motivation for this project comes from the desire to understand how both neural networks and classical machine learning methods work, by studying a product that comes from both branches of machine learning. Both of these areas are left untouched in the current study plan of the Computer Engineering Degree, so this presents a unique opportunity to dive into a field that is and will be crucial for research.

The starting point of the project comes from a tutorial on Unsupervised Feature Learning and Deep Learning from the University of Standford [1]

1.4 Structure of the document

This document is separated into three big blocks.

The first one is comprised of the theoretical sections that are needed in order to understand Reconstruction ICA, the main focus of the project. This block includes

sections 2 to 6. Here we will look at a brief overview of the different techniques that lead to, or are related to, Reconstruction ICA.

The second block is formed by section 7, which dives into Reconstruction ICA and its relationships to the other parts of the project that lead up to it.

The third block is comprised of the practical side of the project. This includes the experiments performed in order to study how Reconstruction ICA fares in a practical scenario. This is included in sections 8 and 9.

Finally, we can find the conclusions and future work in section 10.

2 Neural Networks

Neural networks are a computational model used in computer science and machine learning, which is inspired in biology. It's based on a large collection of connected simple units, resembling the axons in the biological brain. The objective is to be able to learn from raw data, by collaboration between the neurons to create output stimuli that can be interpreted and used in different ways. These models can be used to approach complex problems such as image classification or voice recognitions, problems that traditional algorithms and computational techniques are not able to handle [2].

2.1 Neuron

A neural network is composed by basic units called neurons. Neurons are interconnected and receive inputs from other units, to produce outputs that are then used as input by another part of the network.

The simplest possible neural network is one comprised of a single neuron. It can be represented as shown in figure 1.

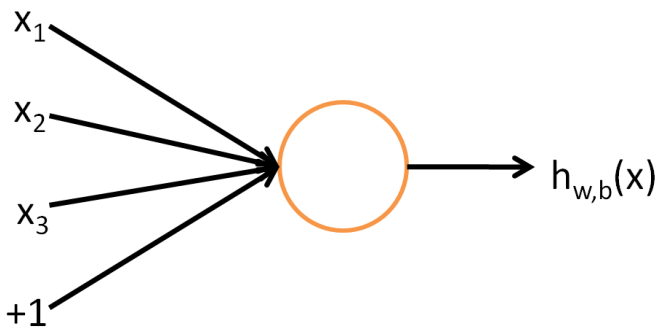

Figure 1: Single neuron.

In this case, we have a neuron that receives 3 input terms x_1, x_2, and x_3 (intercept), and outputs $h_{w,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$, where:

- f is the activation function. It can be defined by several ways depending on the application of the network, we will look into some of them later.
- W and b are the parameters of the network. W is also referred to as the weight matrix, and will change during the training process of the network, as it learns to "solve" the given problem.

2.2 Neural Network model

A full neural network is created by connecting together several simple neurons, in such a way that the output of one of them is used as the input of one (or more) neurons. This creates a structure like the one shown below, where we use circles to represent neurons and arrows to show where each output is used as input. We represent the input to the network as circles too which, in this example, correspond to the first layer L_1 and are considered the input layer. The rightmost layer is the output layer. This is what is observed at the end of the process, in this case, it's a

single output unit. The middle layer, L_2, is a hidden layer, as it's output is not part of the training data and can not be directly observed.

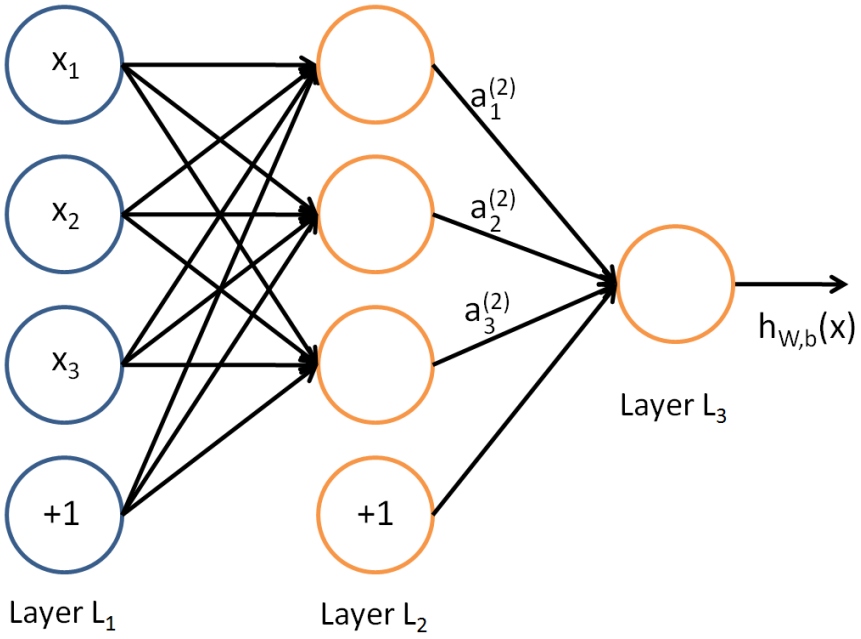

Figure 2: Full network.

Figure 2 shows an example of a simple network. In practical applications (such as the ones we will see in this project), neural networks will be formed by many layers each comprised of many units.

Let n_l denote the number of layers in a neural network, such that n_1 is the input layer and n_{n_l} is the output layer. The network has parameters (W, b) where $W_{ij}^{(l)}$ is the weight of the connection between unit j in layer l and unit i in layer $l+1$, and $b_i^{(l)}$ is the bias in the unit i in layer $i+1$.

Let $a_i^{(l)}$ denote the output value of unit i in layer l. We will call this the activation unit. In the case of the first layer, we will use $a_l^{(1)} = x_i$ to denote the i-th input of the training data.

2.2.1 Forward propagation

An algorithm named forward propagation is used to calculate the activation units, we will take a look at how to calculate those in the following example.

We already know that $a_i^{(1)} = x_i$.

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_1 1^{(2)} a_1^{(2)} + W_1 2^{(2)} a_2^{(2)} + W_1 3^{(2)} a_3^{(2)} + b_1^{(2)})$$

The following equation shows the same calculation using matrices.

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

2.3 Activation functions

Activations functions (f in our examples) are applied to the inputs of the neurons to create an output.

The most common activation functions are the following (See figure 3 for an illustration):

- Linear: The linear function is defined by

$$f : \mathbb{R} \to \mathbb{R}$$

$$f(x) = x$$

  where k is a constant. This is the most simple activation function and is obviously C^∞.

- Threshold function: The threshold (or step) function is defined by:

$$f : \mathbb{R} \to \{0, 1\}$$

$$f(x) = \begin{cases} 1 & \text{iff } x > 0 \\ 0 & \text{iff } x <= 0 \end{cases} \tag{1}$$

  This function is non-differentiable in $x = 0$ so we can not apply gradient descent, but it can still be used in simple neural networks.

- Rectified linear: The rectified linear function is defined by:

$$f : \mathbb{R} \to [0, \infty)$$

$$f(x) = max(0, x)$$

  Similarly to the threshold function, this function isn't non-differentiable and can not be used with gradient descent.

- Sigmoid: The sigmoid function is defined by:

$$f : \mathbb{R} \to (0, 1)$$

$$f(x) = \frac{1}{1 + e^{-x}}$$

  This function is C^∞ and

$$f'(x) = f(x)(1 - f(x))$$

- Hyperbolic tangent: The hyperbolic tangent function is defined by:

$$f : \mathbb{R} \to (-1, 1)$$

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

  This function is C^∞ and

$$f'(x) = 1 - f^2(x)$$

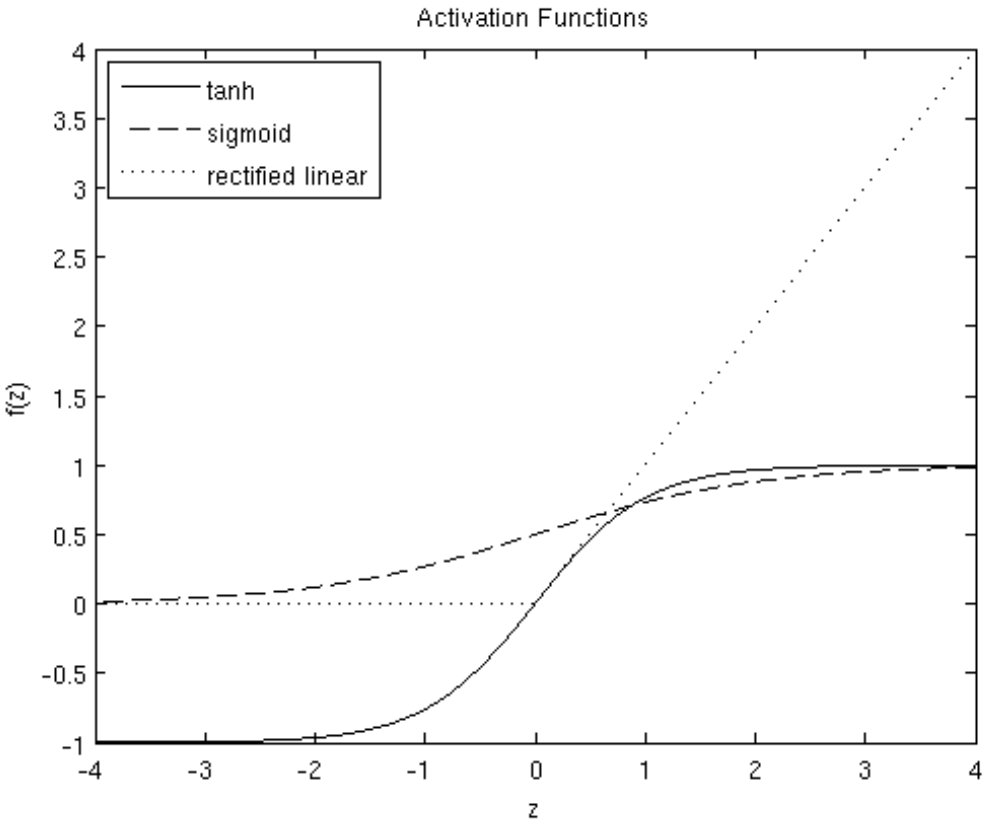

Figure 3: Activation functions.

2.4 Main types of neural networks

There are several types of neural networks, each of which have their own structures and characteristics. Neural networks are trained with a data input, which we call training set. Depending on how the task and the training set, we can differentiate two different types of neural networks: supervised and unsupervised networks.

2.4.1 Supervised neural networks

In supervised learning, the network is fed with a training set which includes the desired output. This allows the network to learn by minimizing the error between the predicted output and the known output. This type of learning is applied to classification and regression problems, the first one deals with classifying data into categories and the latter with predicting the results of a continuous function.

2.4.2 Unsupervised neural networks

Unsupervised learning deals with a training set that does not know the desired output. This means that the network has to learn by using only the input data, by looking for structures in the data and working with functions created by the programmer.

2.5 Learning in a neural network

When we talk about learning in neural networks we refer to the process of modifying the weight values by using the backpropagation algorithm [3].

The parameters are found by optimizing functions that represent the error between the outputs. These functions are called cost functions, and can be optimized using methods such as gradient descent [4].

The cost function represents the error between the output and the desired output. The objective is to find parameters such that this error is the closest to zero as possible.

3 Autoencoders

3.1 Introduction

Autoencoders are one type of neural network that focus on compressing and decompressing data automatically. They work by applying backpropagation looking for outputs that are equal to the inputs, i.e, $y^{(i)} = x^{(i)}$. In figure 4 we can see the structure of an autoencoder, where $x_i = \hat{x}_i$.

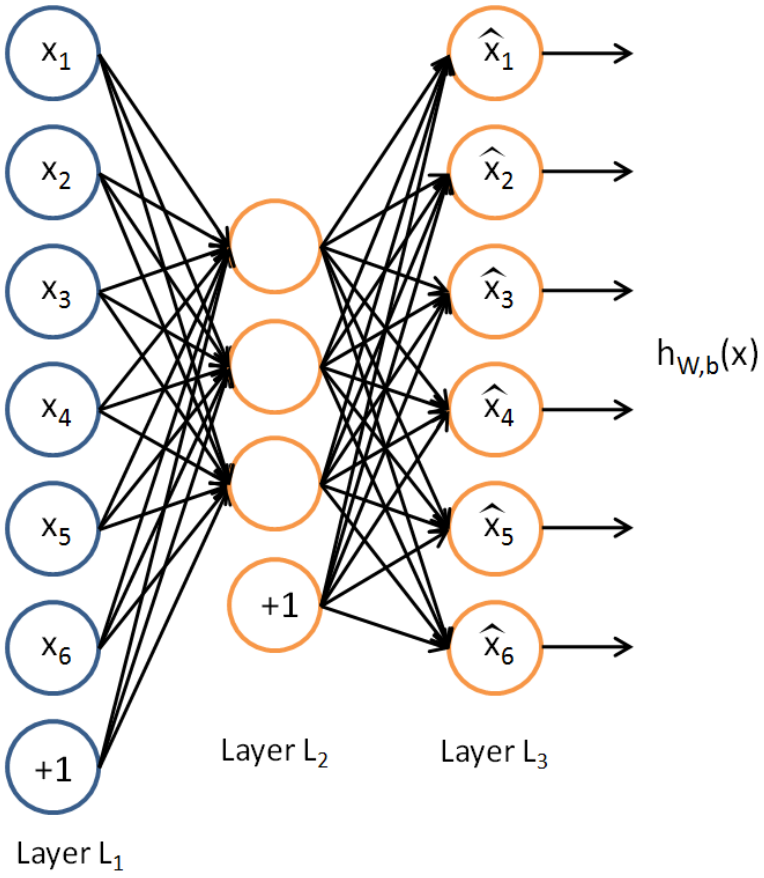

Figure 4: Autoencoder.

3.2 Definition

An autoencoder is separated in two parts, the encoder and the decoder. The simplest case is defined by the encoder taking an input $x \in \mathbb{R}^d$ and mapping it to $z \in \mathbb{R}^p$ such that $z = f(Wx + b)$, f is an activation function, W is a weight matrix and b is a bias vector. The decoding step maps z to the reconstruction $\hat{x} \in \mathbb{R}^d$, by applying $\hat{x} = f'(W'z + b')$.

The objective of an autoencoder is to learn an approximation to the identity function, such that the output $\hat{x}_i$ is similar to x_i ($h_{W,b}(x) \approx x$). At first this may seem a trivial problem, as simply applying the identity function, i.e. $f(x) = x$, would work. The way we use the autoencoder is by applying constraints to the network. For instance, we could limit the number of neurons, or the number of layers. This way, the network will have to learn an efficient way to store the information with

less amount of space, so that it can be restored to be close to the input. This can be seen as an application of compressing an decompressing data, with losing as few information as possible. Taking a look at the parameters learned, we can also discover interesting information about the structure of the data.

3.3 Interpretation

Lets take a look at an example of autoencoders applied to image data. Consider a dataset of images representing hand-written digits, of size 28x28 pixels [5]. Examples of 4 random digits are shown in figure 5.

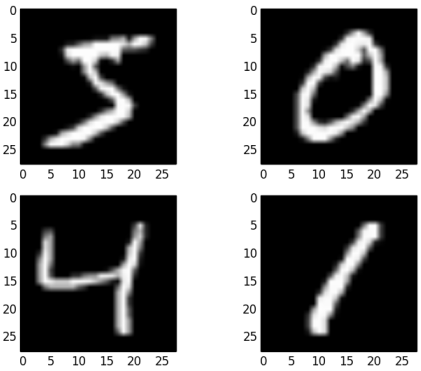

Figure 5: Example of images from the dataset.

Then our input layer would have 784 units, one for each pixel. If we set up an autoencoder with one hidden layer with 200 units, we are forcing the network to learn a way to compress the data such that it can be recreated as well as possible. If the data isn't random, the parameters learned represent how the data is structured. In figure 6 we can see the visualization of the vectors of W after training an autoencoder with 200 hidden units.

In this case, and in the case of working with image data, we end up with a series of "pen-strokes", which show the underlying structure of the original images. This can be interpreted as a series of edge detectors. Therefore, by combining the elements of this basis we can go back to the original data, which should be close to the original image. How close it is to the original depends on many factors, such as the correlation present in the original images or the constraint we have applied to the network. We optimize the representation by looking for the parameters that will limit the error in the reconstruction fase, i.e, those that produce a $\hat{x_i}$ closer to x_i.

In this example we have looked at how limiting the number of units can help us identify patterns in the data. But autoencoders can also be used to learn overcomplete representations of the data as we will explore in the next section. This means having a representation with a dimension higher than that of the original data.

These features learned and the new representations are helpful when facing problems of recognition and classification when dealing with images, audio, and other input types.

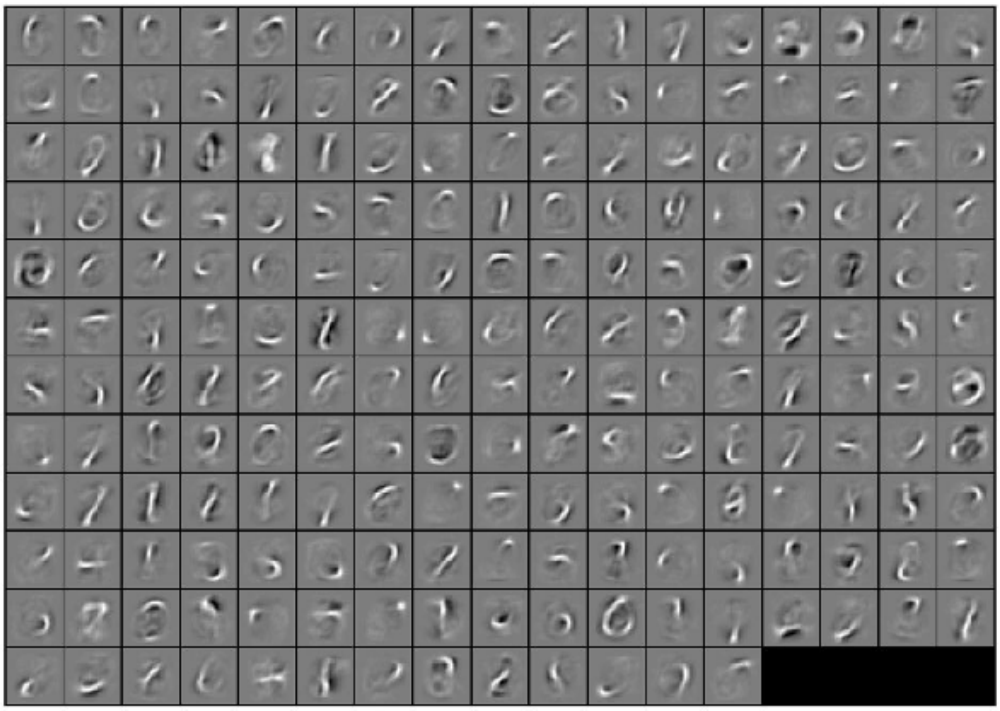

Figure 6: W extracted from the autoencoder [6].

4 Sparsity and Sparse Coding

Techniques such as Principal Component Analysis (PCA) allow us to represent data efficiently by finding a basis of less dimensions in which to represent the data. But sometimes, we wish to learn an over-complete basis in which the dimensions of the output will be bigger than the input. This can be useful to better capture structures and patterns inherent in the original data in some applications [11].

Sparse coding is a type of unsupervised method that is used to learn sets of over-complete bases to represent data. The objective is to represent an input x as linear combination of elements of the base ϕ_i, such that:

$$x = \sum_{i=1}^{k} a_i \phi_i$$

where $x \in \mathbb{R}^n$ such that $k > n$.

Due to $k > n$ the coefficients a_i aren't uniquely determined, so we have to add a new criterion to avoid using degenerate basis. For this purpose, we introduce the concept of sparsity.

By sparsity we refer to having as few components a_i away from zero as possible. This means that we want most of the a_i components to be 0, or close to 0. In the case of image data this idea comes motivated in by the fact that most images can be described by the superposition of elements such as edges or surfaces.

The general optimization function of a sparse coding network is:

$$min_{a_i^{(j)},\phi_i} \sum_{j=1}^{m} ||x^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i||^2 + \lambda \sum_{i=1}^{k} S(a_i^{(j)}) \tag{2}$$

where m is the number of vectors or samples and $S(\cdot)$ is the sparsity function that penalizes a_i for being far from zero.

If we look at the above function as two different parts, we can see that:

$$\sum_{j=1}^{m} ||x^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i||^2$$

is the reconstruction term that forces the algorithm to look for a good representation of x in the new basis, by minimizing the difference between x and the representation in the new basis.

The second term:

$$\sum_{i=1}^{k} S(a_i^{(j)})$$

is the sparsity penalty which forces the representation to be sparse. S can be defined in several ways as we will see next. A constant (λ in this case) is used to determine the relative importance of the two terms when applying the optimization method. In a practical scenario, λ can be varied to analyze results for different values of it.

The current formulation has a problem which becomes present when we see that we can make the sparsity penalty arbitrarily small when we make a_i very small and scale ϕ_i to a large constant. To prevent this, we can add a norm restriction such

that it prevents ϕ_i becoming too large. The sparsity cost with the new restriction is therefore:

$$min_{a_i^{(j)},\phi_i} \sum_{j=1}^{m} ||x^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i||^2 + \lambda \sum_{i=1}^{k} S(a_i^{(j)})$$
$$\text{such that} \quad ||\phi_i||^2 \leq C \quad \forall i = 1, ..., k$$

4.1 Sparsity Norms

The most simple sparsity norm is the L_0 norm. This norm is defined as $S(a_i) = 1$ when $|a_i| > 0$ and 0 when $|a_i| = 0$. This is the most straightforward norm, but it values the same for the cases when a_i is very close to 0 and when it's very far away from it. This can lead to inaccuracies when dealing with complex images, as the linear combinations will not be that straightforward. A common norm is the L_1 norm, defined as $S(a_i) = |a_i|_1 = \sum_{j=1}^{n} |x_i^{(j)}|$. Another common function is the log penalty, defined by $S(a_i) = log(1+a_i^2)$. Notice that both of these functions take into account how far away the coefficients are from 0, therefore optimizing the function better. These functions are also differentiable, which the L_0 norm isn't. This allows the use of gradient descent for the optimization.

5 Principal Component Analysis

5.1 Introduction

Principal Component Analysis (PCA) is an algorithm used to reduce dimensions of data. This technique can be used to speed up the use of algorithms by using a representation of the original data that's smaller than the original. It's can be helpful when dealing with highly correlated data, such as is the case of images, where nearby pixels are usually very similar to each other. This allows a smaller representation in terms of dimension of the original image, without losing too much information.

5.2 Definition

Let X be our data matrix, with n rows and d columns, where n is the number of samples and d is the number of features (i.e. pixels in the case of images). Before applying PCA, the data has to be pre-processed so that it has zero empirical mean for each of the columns (features).

Mathematically speaking, PCA is an orthogonal linear transformation that changes the data matrix X to a new coordinate system. The properties of this transformation and the new coordinate system make it so that the first coordinate contains the greatest variance by some projection of the original data. This coordinate is called the first principal component. Subsequently, the second greatest variance lies in the second coordinate, and so on.

The transformation is defined by a set of vectors of weights $\{w_{(k)}\} = (w_1, ..., w_m)_{(i)}$ that maps each row of X to a new basis $\{t_{(i)}\} = (t_1, ..., t_m)_{(i)}$ such that for each row $x_{(i)}$ of X, $t_{k(i)} = x_{(i)} \cdot w_{(k)}$ for $i = 1, ..., n$ and $k = 1, ..., m$ such that the first variable of T inherits the maximum possible variance from X, with the restriction that each vector w is a unit vector. In matrix form we get the decomposition shown in equation 3.

$$X = WT \tag{3}$$

Where $X \in \mathbb{R}^{d \times x}$, $W \in \mathbb{R}^{d \times l}$, and $T \in \mathbb{R}^{l \times n}$ (figure 7).

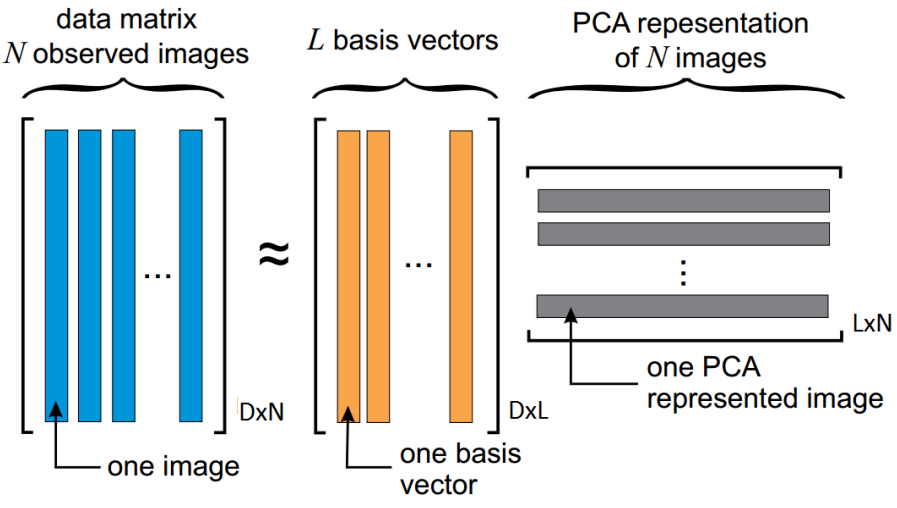

Figure 7: PCA matrix representation.

5.3 Applications

PCA is the basis on which the concept of whitening [7] is based upon. The goal of whitening is to make the input less redundant, such that we achieve a representation in which the features are less correlated and the features have the same variance.

The effects of whitening can be seen in the following example.

In figure 8 we can see original images of random patches extracted from a natural image. The borders are faded out and are not very clear, except for some cases.

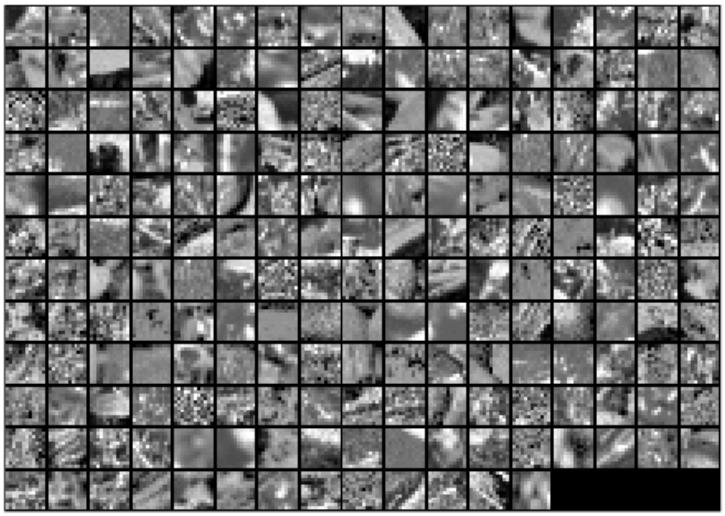

Figure 8: Image patches before whitening.

In figure 9, we can see the effects of whitening. The edges are now more visible stand out more. Whitened images have been proven to be more effective than natural images when dealing with image recognition classification tasks, and are used as a preprocessing step in many algorithms such as Independent Component Analysis (ICA) [8].

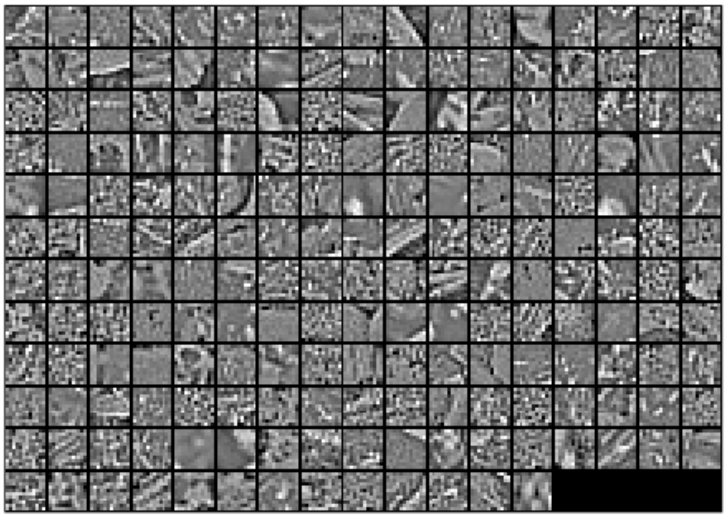

Figure 9: Imatge patches after whitening.

6 Independent Component Analysis

6.1 Introduction

Independent component analysis (ICA) is a statistical method that is used to reveal hidden factors that underlie sets of data. This data can represent sets of random variables, images, or signals. ICA defines a model for the observed data, which are assumed to be linear mixtures of unknown variables mixed with an unknown mixing system. The unknown variables are the independent components of the observed data, and can be extracted using ICA.

6.2 Definition

The ICA model is defined by the following equation:

$$x = As$$

where x is the vector consisting of the mixtures (or original data) (x_1,...,x_n), A is the mixing matrix, and s is the vector corresponding to the independent sources s_1,...,s_n.

The ICA model describes how the observed data x is defined by mixing the components s. Given the observed data x we can not directly extract the sources s, nor know the mixing matrix A. Therefore, we must estimate both A and s from the original data x. Note that finding A or s will in turn give us the other one by applying matrix properties.

Let W be the inverse of A, then the model can be rewritten as:

$$s = Wx$$

Another interpretation is as follows: Given some raw data x , the objective is to find a set of vectors (which will be the columns of our matrix W) that will make the features s sparse; while being an orthonormal basis. (An orthonormal basis is a basis (x_1,...,x_n) such that $x_i \cdot x_j = 0$ if $i \neq j$ and $x_i \cdot x_j = 1$ if $i = j$). In other words, our matrix W will map raw data x to features s. We can define this idea as an optimization problem:

$$\min_{W} \quad f(Wx) \text{ such that } WW^T = I$$

where f is a non-linear convex function and $W \in \mathbb{R}^{k \times n}$ (where k is the number of features (components) and n the amount of data vectors in x). The orthonormality constraint $WW^T = I$ is used to prevent degenerate basis.

6.3 Algorithm: FastICA

6.3.1 FastICA for single component extraction

FastICA [9] is an iterative algorithm that looks for the direction of the weight vector $w \in \mathbb{R}^N$ that maximizes the non-Gaussianity of $w^T X$ (with $X \in \mathbb{R}^{N \times M}$).

The algorithm works by initializing a random weight and iterating until there is convergence. g is a nonquadratic nonlinear function that measures non-Gaussianity. Given a random w as a staring point:

1. $w^+ \leftarrow E\{Xg(w^TX)^T\} - E\{g'(w^TX)\}w$
2. $w \leftarrow w^+/||w^+||$
3. If not converged (old w and new w do not point in the same direction), back to 1

6.3.2 FastICA for multiple component extraction

When estimating multiple components, they have to be mutually independent. To achieve this, we run the single unit FastICA using several units with weight vectors $w_1, ..., w_n$. To avoid having multiple vectors converging to the same maxima, the outputs $w_1^T x, ..., w_n^T x$ have to be decorrelated.

6.4 Interpretation

6.4.1 ICA on images

When dealing with images, we try to look for an expression $\lambda_1 s_1 + \lambda_2 s_2 + ... + \lambda_n s_n$ such that the coefficients are independent, i.e. that the coefficient of one gives the minimum possible information about the coefficients of the others.

When working with big images, ICA can be hard to apply to the full image, so smaller random patches are extracted and ICA is performed on them. For example, when applying ICA on patches extracted randomly from the image in figure 10 we end up with a matrix of weights W that, when we display each column as if it where an image, we get the representation shown in figure 11.

Figure 10: Natural image.

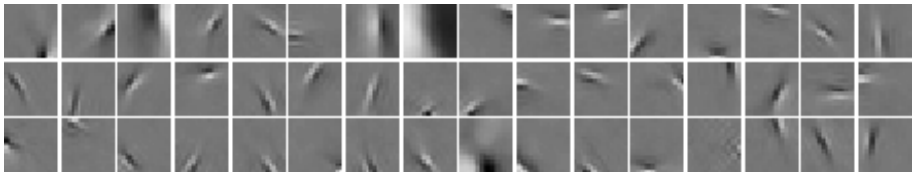

Figure 11: ICA on patches.

Therefore, the reconstruction we are looking for consists of combining each of the features extracted such that we achieve the original image, with having the coefficients be as independent as possible. Generally speaking, features extracted with ICA represent edges or pen-strokes.

6.4.2 ICA on signals

In the case of signals, the interpretation and what we are looking for is different. In the case of images, we were interested in the weight matrix W. When dealing with signals, we are interested in the independent sources s_i. In the following example, we can see how given two recordings of the same sound, created by two independent sources illustrated in figure 12 we can find the original sources seen in figure 13 by applying ICA.

This technique can be used with many types of signals. With sound waves for example, we can extract individual speakers from recordings of the speakers speaking at the same time. In the case of electroencephalography, we can extract independent sources of recorded electrical activity in the brain.

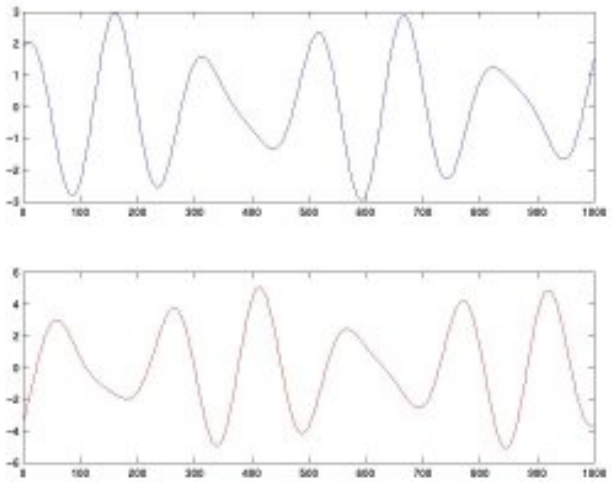

Figure 12: Mixed signals.

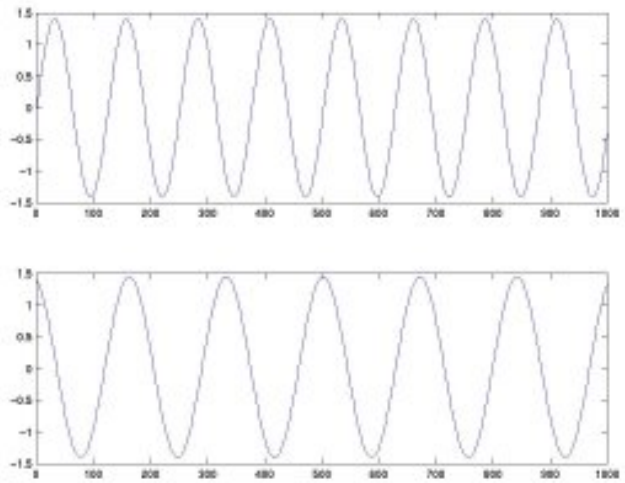

Figure 13: Signals extracted using FastICA.

6.5 ICA drawbacks

As discussed previously, Independent Component Analysis (ICA) and its variants have been used for feature extraction in images successfully. However, ICA has two major drawbacks when dealing with high dimensional data, which in our case means images.
First and foremost, ICA isn't able to learn overcomplete feature representations of data. This means that the number of features extracted by ICA can not be bigger than the data's original dimension. In the case of feature extraction in images and image recognition, sparsity has been proven to work well with classification algorithms such as K-means [10] and RBMs [11]. Moreover, ICA is sensitive to whitening (see PCA whitening). This makes ICA difficult to scale to high dimensional data. Finally, this problem has no simple analytic solution, and is costly to optimize using gradient descent, as every iteration has to be followed by an extra step that maps the basis to the space of orthonormal basis.

These drawbacks arise from the orthogonality condition required in the ICA formulation: $WW^T = I$. This constraint can not be satisfied if the dimensions of the features extracted extend those of the original data.

7 Reconstruction ICA

7.1 Introduction

A number of algorithms based on sparsity have been shown to work well for learning feature representations that can be used to help with object recognition. These includes algorithms such as sparse-autoencoders, sparse coding, Restricted Boltzmann Machines or Independent Component Analysis. ICA in particular has been shown to work well when dealing with images and has been used to learn features that achieved state-of-the-art performance when applied to object recognition tasks. Reconstruction ICA (RICA) was created to overcome the drawbacks of ICA present due the orthogonality condition present in the formulation of ICA.

7.2 Definition

RICA works by replacing the orthogonality condition $WW^T = I$ present in ICA with a soft reconstruction penalty, similar to how sparse coding and sparse autoencoders use reconstruction terms.

The optimization problem defined by RICA is:

$$\min_{W} \quad \lambda||Wx||_1 + \frac{1}{2}||W^TWx - x||_2^2 \tag{4}$$

RICA provides three major benefits over using ICA:

1. The computational cost of the optimization problem is reduced as RICA removes the need of using a constrained optimizer. Instead, we can use existing unconstrained solvers such as L-BFGS or CG that result in fast convergence.

2. RICA allows the extraction of overcomplete featyres, something standard ICA can not learn due to the constraint present in ICA, $WW^T = I$. Other work has shown benefits of using overcompleate features in tasks such as object recognition. CITA.

3. RICA is less sensitive to whitening. When using ICA, the data has to be whitened but this is a problem with high computational cost when dealing with a large number of features, something that happens with big images. RICA can handle data with approximate whitening or even without whitening [12].

7.3 Implementation

In order to implement gradient descent we need to compute the gradient of the cost function. To do so, we will derive the gradient by looking at each part of the function separately. Remember that W refers to the weight term and x to the raw data (3).

First, we will derive the gradient of $||W_x||_1$. To implement the L1 regularization, we will use the L1-norm defined by $f(x) = \sqrt{x^2 + \epsilon}$. This is straightforward:

$$\Delta_W(\sqrt{(Wx)^2 + \epsilon}) = \frac{1}{2\sqrt{(Wx)^2}} 2Wxx^T = \frac{Wxx^T}{\sqrt{(Wx)^2}}$$

We defined the reconstruction term as $||W^T W x - x||_2^2$. We can derive the gradient of this term by using the backpropagation idea. Let's interpret the reconstruction term as the neural network illustrated in figure 14.

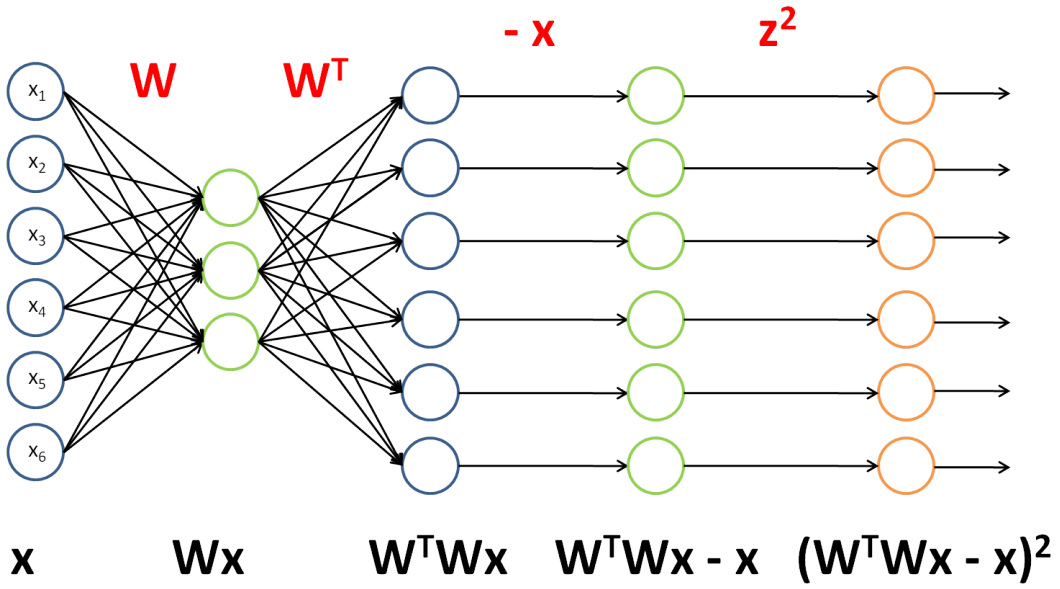

Figure 14: Reconstruction term as a neural network [14].

Using this interpretation, we can calculate its gradient using tables 1 and 2.

Layer	Weight	Activation function
1	W	$f(z_i) = z_i$
2	W^T	$f(z_i) = z_i$
3	I	$f(z_i) = z_i - x_i$
4	N/A	$f(z_i) = z_i^2$

Table 1: Activation functions of each layer.

Layer	Derivative of Activation function	Delta	z
4	$f'(z_i) = 2z_i$	$f'(z_i) = 2z_i$	$W^T W x - x$
3	$f'(z_i) = 1$	$(I^T \delta^{(4)}) \cdot 1$	$W^T W x W$
2	$f'(z_i) = 1$	$((W^T)^T \delta^{(3)}) \cdot 1$	$W x W$
1	$f'(z_i) = 1$	$(W^T \delta^{(2)}) \cdot 1$	x

Table 2: Derivatives of the activations and δ of each layer.

We have to find the gradient with respect to each instant of W in the network, in this case, W^T (equations (5)) and W (equations (6)):

$$\begin{aligned} \Delta_{W^T} ||W^T W x - x||_2^2 &= \delta^{(3)} a^{(2)T} = 2(W^T W x - x)(W x)^T \\ \Delta_W ||W^T W x - x||_2^2 &= \delta^{(2)} a^{(1)T} = (W)(2(W^T W x - x)) x^T \end{aligned} \tag{5}$$

$$\begin{aligned} \Delta_W ||W^T W x - x||_2^2 &= \Delta_{W^T} ||W^T W x - x||_2^2 + \Delta_W ||W^T W x - x||_2^2 \\ \Delta_W ||W^T W x - x||_2^2 &= (W)(2(W^T W x - x)) x^T + 2(W x)(W^T W x - x)^T \end{aligned} \tag{6}$$

When combining every derivation, we get that the gradient of the optimization problem

$$\min_W \quad \lambda||Wx||_1 + \frac{1}{2}||W^TWx - x||_2^2$$

is

$$\lambda\frac{1}{2\sqrt{(Wx)^2}}2Wxx^T + \frac{1}{2}(W)(2(W^TWx - x))x^T + 2(Wx)(W^TWx - x)^T$$

7.4 Relationship with Autoencoders and Sparse Coding

ICA has been linked to sparse coding and sparse autoencoders because they all learn edge filters and detectors when dealing with natural image data. As we have seen, their formal definitions are very similar (equations (2) and (4)), and, under certain conditions, they are mathematically equivalent.

The main difference between ICA and sprase coding and autoencoders is the use of the hard orthonormality constraint. We can see that, when the data $\{x^{(i)}\}_{i=1}^m$ has zero mean, we can derive the RICA reconstruction cost from the ICA orthonormality constraint.

The following lemmas [10] are used to show the relationship between them. We use $||\cdot||_2$ to denote the L_2 norm and $||\cdot||_F$ to denote the Frobenius norm.

Lemma 1: If the original data $\{x^{(i)}\}_{i=1}^m$ is whitened, the orthonormality cost is equal to the RICA reconstruction cost, i.e:

$$\lambda||W^TW - I||_F^2 = \frac{\lambda}{m}\sum_{i=1}^m ||W^TWx^{(i)} - x^{(i)}||_2^2$$

Lemma 2: The column orthonormality cost is equivalent to the row orthonormality cost

$$\lambda||W^TW - I_n||_F^2 = \lambda||WW^T - I_k||_F^2 + c$$

where c is a constant.

Given these lemmas, we can extract the following conclusions:

1. RICA is equal to ICA for complete or undercomplete representations of the original data if λ approaches infinity and the data is whitened. The RICA formulation:

   $$\min_W \frac{\lambda}{m}\sum_{i=1}^m ||W^TWx^{(i)} - x^{(i)}||_2^2 + ||g(Wx)||_1$$

   is, using the lemmas above, equivalent to:

   $$\min_W \lambda||W^TW - I||_F^2 + ||f(Wx)||_1 \quad \text{and}$$

   $$\min_W \lambda||WW^T - I||_F^2 + ||f(Wx)||_1$$

   Therefore, when λ approaches infinity, the orthonormality constraint becomes strict, therefore we have the following optimization problem:

   $$\min_W ||f(Wx)||_1 \text{ such that } WW^T = I$$

   which corresponds to the conventional ICA formulation.

2. RICA is equal to a sparse autoencoder when we set the activation function $\sigma(x)$ to be a linear function (or identity function), we use the soft L_1 sparsity function for the activations and set the bias terms to 0.

3. RICA is equal to the sparse coding formulation when we ignore the maximum norm constraint and set $x^{(j)} := Wx^{(j)}$.

8 Programming and libraries

8.1 MATLAB

MATLAB is a high level computing environment and programming language for the development of algorithms, numerical computation, data analysis and visualization, and other mathematically based functions. MATLAB allows for faster computation when compared to traditional languages such as C++ when dealing with matrix operations and other high cost functions due to how the language was developed to optimize vector and matrix computations.

During the experimentation part of the project we will be dealing with large amounts of data (images and signals) which will be represented computationally as matrices. As we have seen, the algorithms we are going to use are based on matrix and vector operations, so MATLAB will be an efficient environment in which to perform our tests. It also provides functions and libraries we are going to use to perform the optimization and classification tasks.

8.2 PCA and ICA Package

To perform both PCA and ICA, we have used the implementation found in the PCA and ICA package by Brian Moore based on Hyvrinen, Aapo, and Erkki Oja. "Independent component analysis: algorithms and applications." To extract the principal components we can use the function:

```
[Zpca, U, mu] = PCA(Z,r);
```

to extract the principal components and the elements needed to find the reduced approximation of the data.

To use the fastICA algorithm we can use the function:

```
[Zica, W, T, mu] = fastICA(Z,r);
```

to extract the independent components and the transformation matrices found in the process.

8.2.1 minFunc

minFunc [14] is a MATLAB function for unconstrained optimization of differentiable real-valued multivariate functions using line-search methods. This function has been used in the implementation of RICA and to optimize all other cost functions that have been used in the project.

To use minFunc, we need the function to optimize (rica), the input data (whitened) and the chosen parameters (number of iterations, λ,...). To apply minFunc we use:

```
[W, cost, ~] = minFunc( @(theta) rica(theta, data, params), randomTheta, options);
```

8.2.2 SoftMax Classifier

To use this classifier in MATLAB, we will use the Neural Network Toolbox released in the R2015a version of MATLAB. The toolbox provides algorithms, models and applications to train, visualize and simulate neural networks.

Example of softmax classifier:

To train a classifier from a data matrix $X \in \mathrm{R}^{mxn}$, where m is the number of features and n is the number of samples, with a target matrix $T \in \mathrm{R}^{rxn}$ where r is the number of target classes for classification, and n is the number of samples, we use:

```
net = trainSoftmaxLayer(X,T);
```

To classify observations $Y \in \mathrm{R}^{mxr}$ given the trained classifier:

```
Z = net(Y);
```

8.2.3 UFLDL MATLAB Modules

The University of Standford has put together a series of functions and scripts to work with machine learning applications as part of their Unsupervised Feature Learning and Deep Learning course [15]. Throughout this project, some functions from the module have been used to help with the tasks, such as:

`display_network.m`

to visualize groups of images stored as columns of a data matrix.

`checkNumericalGradient.m`

to check gradient implementations of the RICA code.

`loadMNISTImages.m and loadMNISTLabels.m`

to load the MNIST data stored as binary files.

9 Experiments

In this section we evaluate the performance of the studied algorithms for feature extraction, PCA, ICA and RICA. The first part consists of a analyzing how the performance of a classification test varies depending on how the features are extracted using RICA. We examined the number of features, the number of iterations, and the value of λ in the RICA formulation. In the second part, we looked at the visual differences between extracting features using PCA, ICA, and RICA.

9.1 Dataset

To perform the tests present in this project we used the MNIST dataset (Modified National Institute of Standards and Technology dataset). This is a database of greyscale images of handwritten digits of size 28 pixels by 28 pixels. This dataset is widely used for research purposes in order to train and test machine learning algorithms and object recognition tasks. See figure 15 for an example of sample digits from the dataset.

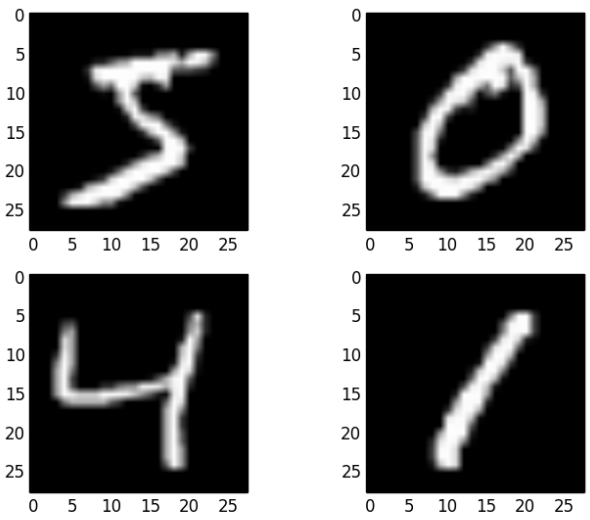

Figure 15: Example of 20 images from the MNIST dataset.

9.2 Classification test

To compare performance between tests we have performed a classification test on the MNIST dataset using a softmax classifier. The objective is not to get the best classification possible, but to compare performance in RICA when varying the number of features extracted, the number of iterations performed, and the value of λ, and to understand how they affect the results. The focus is not on how accurate we can get the classification but on how the accuracy varies depending on how we represent the data.

9.2.1 Softmax classifier

The softmax classifier is a generalization of logistic regression. Logistic regression is a binary model, which represents a function that given an input x, outputs a binary prediction (only 2 possible outputs). The softmax classifier uses the softmax function in equation (7) to assign probabilities to n categories and converts the logistic regression into a multinomial logistic regression that allows classification of more than two categories. This classifier was used to compare performance within the tests.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{7}$$

9.2.2 Performance tests

The train and test data is comprised of images from the MNIST dataset, 60000 images for the train dataset and 10000 images for the test dataset. The images from the test set are not be part of the train set.

The structure of the tests is as follows:

1. Load train and test data
2. Pre-process train and test data. The data was whitened using PCA Whitening techniques [6].
3. Train RICA with the train data
4. Change train data to new basis found with RICA
5. Train classifier with train data in this new basis
6. Change test data to new basis found with RICA
7. Test classifier performance with test data in the new basis

9.3 RICA parameter analysis

As we have seen, RICA can be used for several tasks such as extracting features from images or finding new basis to help with classification tasks. When considering applying RICA to a problem, paratmeters have to be decided and decisions have to be made, both when implementing the solution and when analysing the problem.

In this section study how varying the parameters changes the result of applying RICA to the MNIST dataset, and, if applicable, how it translates in terms of accuracy when applying a predictive model to the new representation of the images in the new basis.

9.3.1 Number of features

The number of features (size of W in equation (3)) determines the dimension of the new representation of the data. This new dimension can be smaller or larger than the original. As we have discussed previously, having an overcomplete set of vectors is exclusive to RICA, and can not be done with conventional ICA. Determining how many features you want is dependent on the data, the task, and the computational power available. Applying RICA can be computationally expensive when the dataset is large, which is the case when dealing with images. This means that compromises must be made to achieve the best representation possible within the technical limitations.

Using the MNIST dataset, we want to explore how the set of basis vectors changes when increasing the number of features extracted with RICA, and how the accuracy of a classification task varies depending on the number of features extracted. To do this, we have extracted 7 different basis in which to represent the original images, 5 undercomplete (25, 50, 100, 200 and 400 features), one full (784 features) and one overcomplete (1000 features). All of these basis have been extracted using the same images as input and the same parameters and functions when applying RICA.

First we will look at the visual differences when looking at the vectors that form the basis change matrix. These vectors are the columns of W in equation (3) and can be visualized by treating each column as the pixel representation of each vector of the basis. These are shown in the following figures (16-22).

In the case of 25 features (figure 16) and 50 features (figure 17), we can see that these numbers of features are not enough to be able to separate the patterns inherent in the images. We can observe how each of the vectors represent one or more than one digits overlapping, which makes it harder to recreate the original images through linear combination of the vectors of W.

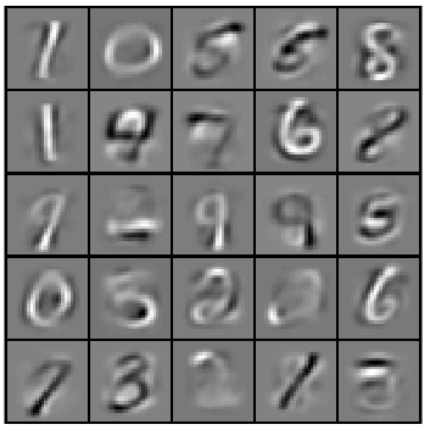

Figure 16: RICA extraction of 25 features.

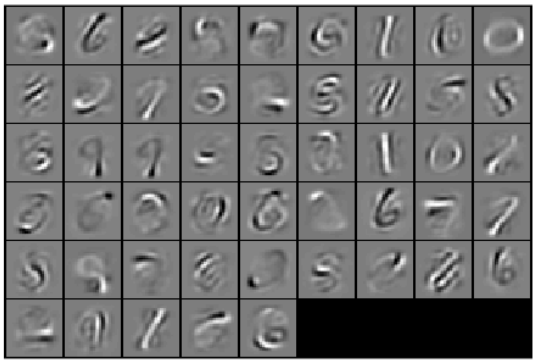

Figure 17: RICA extraction of 50 features.

When the number of features reached 100 (figure 18) and 200 (figure 19), we started to see clearer representations of edges, which is what we were looking for. In the case of 100 we still had some vectors which were not clear, but in the case off 200 most of them were sharper and clearly showed exact edges, instead of a series of overlaps.

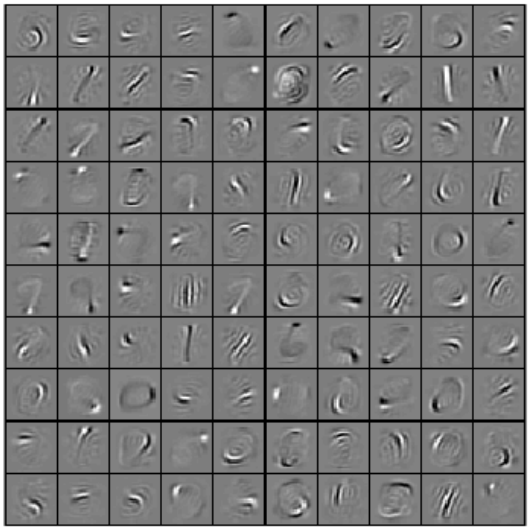

Figure 18: RICA extraction of 100 features.

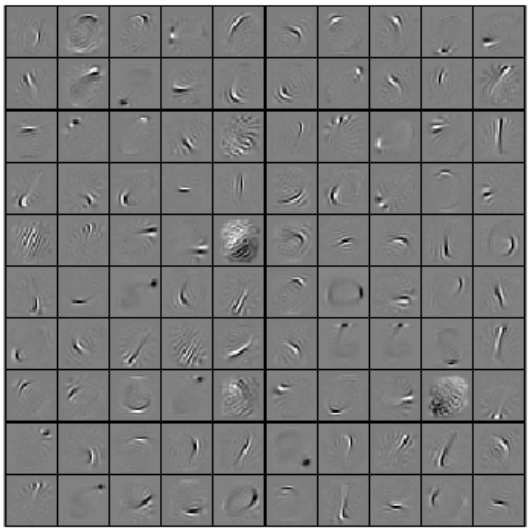

Figure 19: First 100 vectors of RICA extraction of 200 features.

Finally, for the cases with higher number of features, we can see how the edges started to become shorter and in some cases ended up as dots. This suggests that there is no need to extract so many features for these images due to their simplicity. We can see this in the case of 400 features (figure 20), 784 features (figure 21) and 1000 features (figure 22). In the two latter cases, we even ended up with vectors of the basis which were all flat and didn't add any useful information.

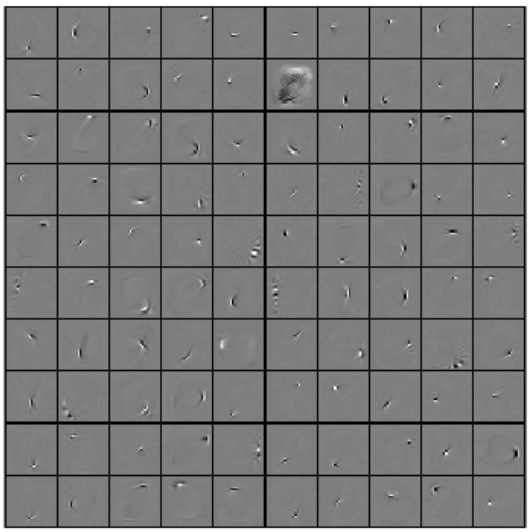

Figure 20: First 100 vectors of RICA extraction of 400 features.

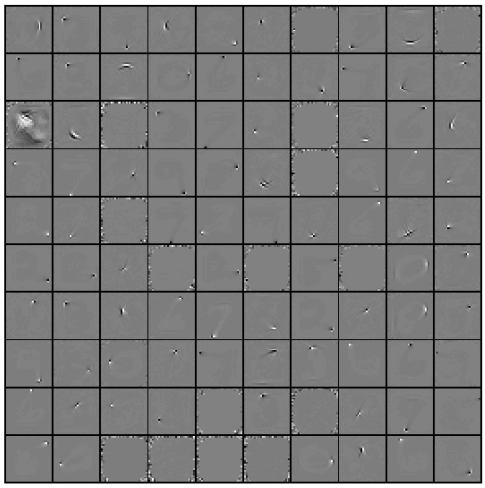

Figure 21: First 100 vectors of RICA extraction of 784 features.

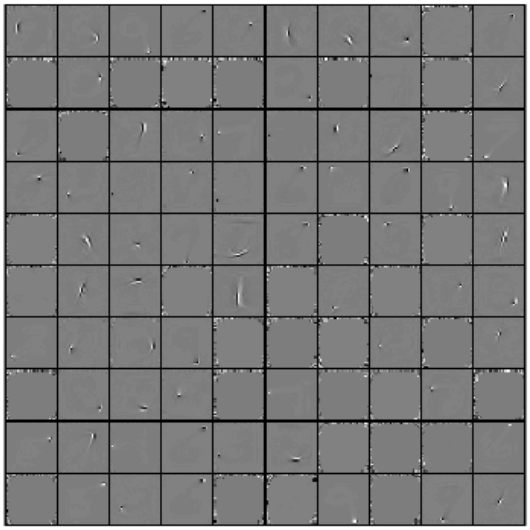

Figure 22: First 100 vectors of RICA extraction of 1000 features.

To test the accuracy when dealing with the new representation, we trained a softmax classifier with the images in the different representations. Then, using a set of test images and after transforming them to the new basis, we tested the accuracy of the classification network. Table 3 and figure 23 summarize the results.

Number of features	Accuracy of test
25	88.5%
50	91.1%
100	92.0%
200	92.8%
400	91.4%
784	88.9%
1000	87.5%

Table 3: Accuracy of tests depending on number of features

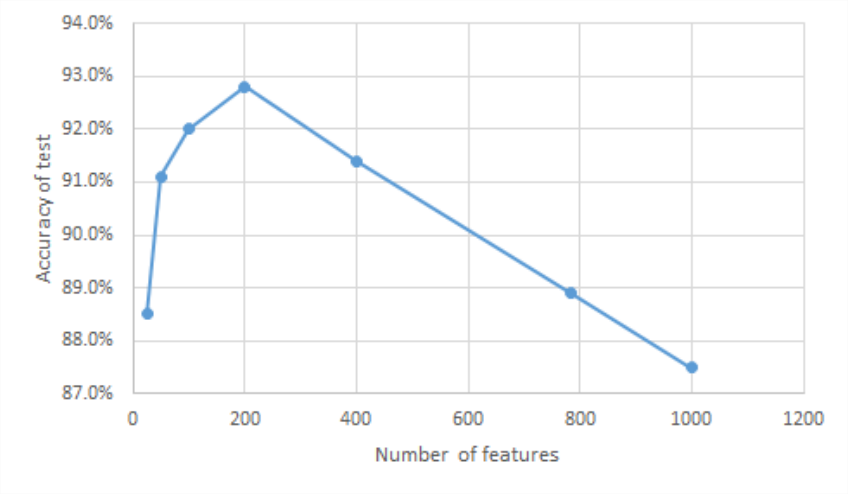

Figure 23: Accuracy of tests depending on number of features.

As we can see, the accuracy increased as the number of features increased until we reached about 200 features. For more than that, the accuracy started decreasing. This indicates that the images were simple enough that they could be represented with less dimensions than their number of pixels and that this new representations gave better results than using more features or even the original images. The full basis and the overcomplete basis probably suffered by overfitting and where not useful for this classification test. Overcomplete basis could be more useful when dealing with larger and more complex images such as fMRI images [16] as the basis would need to be more complex in order to represent the images accurately.

9.3.2 Number of iterations

Whenever we work with a cost function we have to apply an optimization method to find the desired output. As we have seen, this is the case in RICA, where we apply gradient descent to find the optimum W (section 6.3). The process of minimizing a function consists of repeating the iterative algorithm looking for convergence, or the minimum error possible. Depending on the project, we can choose the stopping criterion: by limiting the number of iterations, looking for when the error value crosses a given threshold, etc. Intuitively we can understand that the higher the number of iterations the more optimum the result will be, but each extra iteration adds computational time to the process which results in longer waits when testing.

In this test, we explored how the performance varies by varying the number of iterations done by RICA before stopping, and how much time it takes. The rest of the variables of the algorithm are kept the same with all the tests, and the number of iterations will be the only thing that varies. We will test with 100, 500, 1000, 5000 and 10000 iterations extracting 100 features. The performance and time taken are summarized in table 4 and in figure 24.

Number of iterations	Accuracy of test	Time in seconds
100	90.3%	44
200	91.7%	91
500	91.9%	215
2000	91.9%	867
5000	92.0%	2207
10000	92.0%	4654

Table 4: Accuracy of tests depending on number of iterations

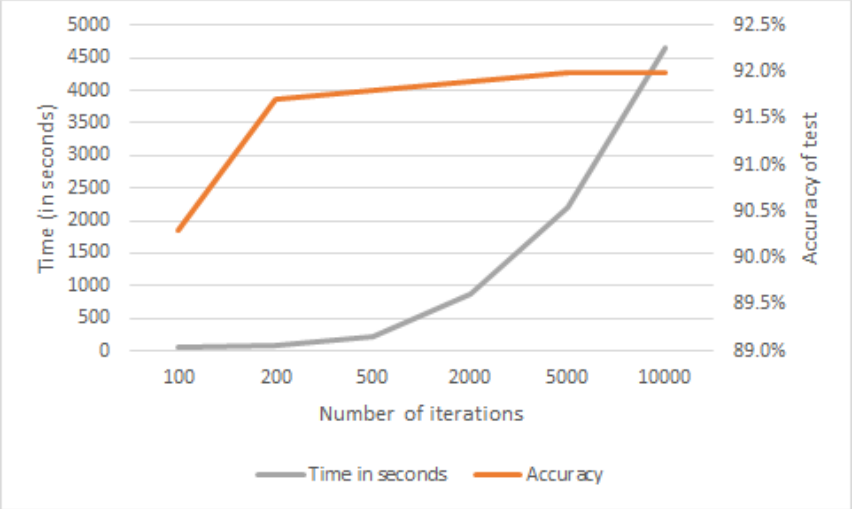

Figure 24: Accuracy of tests depending on number of iterations.

From this test, we can clearly see that there is a point in which the representation is not going to get any better. In this case, the increase in accuracy going from 500 to 10000 is only 0.1%. This reinforces the concept that this application of RICA is a fast converging one.

9.3.3 Value of λ

As described in the formulation of RICA in equation (3), there are two parts to the optimization problem.

$$\lambda||Wx||_1 + ||W^TWx - x||_2^2$$

The first term deals with the sparsity constraint, while the second one looks for a accurate recreation of the original image, by looking for it to be close to the recreation W^TWx. The parameter λ defines how much weight we want to give to the sparsity constraint in relation to the recreation condition. The more weight we give to the sparsity constraint the less precise will the recreation be, and vice-versa. In table 5 and figure 25 we summarize the accuracy of the classification test when varying the value of λ. For these tests, we extracted 100 features with 500 iterations on RICA.

λ	Accuracy of test
0	91.8%
5×10^{-5}	91.8%
5×10^{-4}	91.9%
5×10^{-3}	89.7%
5×10^{-2}	66.7%
5×10^{-1}	41.2%

Table 5: Accuracy of tests in relation to λ

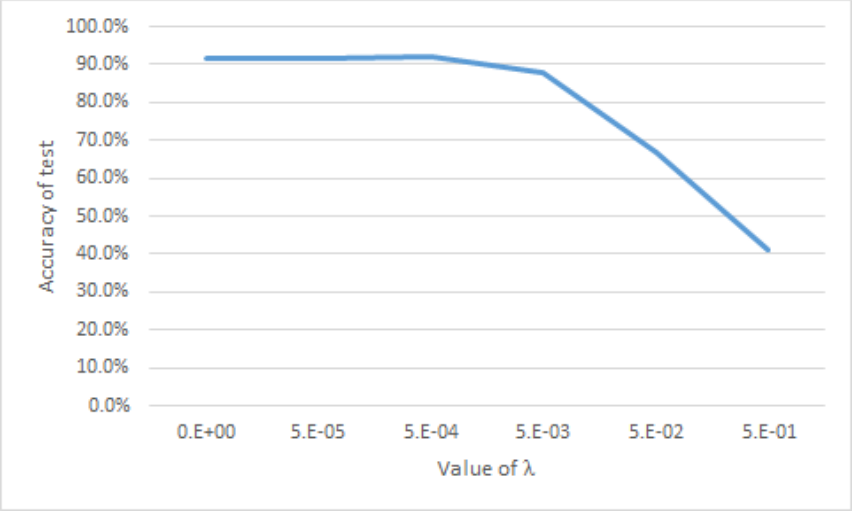

Figure 25: Accuracy of tests depending on number of iterations.

From these results, we can reach the conclusion that the recreation constraint of the RICA formulation is the most important part and should have the bigger weight when optimizing the function in order to find W. However, having no sparsity constraint did not produce the best results. In fact, we have seen that having a small sparsity constraint can help prevent degeneration of the basis and thus give better performance.

9.4 Comparison of PCA, ICA and RICA

In the previous section we have applied RICA to complete images of 28x28 pixels. Unfortunately, as has been explained in section 5.6, ICA requires high computational power and time to be applied to big datasets of high-dimensional data. This is usually the case when dealing with images, even so with small images like the ones used previously. Due to the available hardware and time we were not able to perform ICA on the full set of MNIST images as we have done with RICA. Even though ICA can not be applied to larger images, it is still tested with small patches of larger images.

A patch is a small $n \times n$ section of an image that is usually extracted randomly from the original dataset. A large number of patches are extracted and ICA can be applied to them as they are smaller. The representations of the patches extracted with ICA can be used when applying other algorithms such as convolutional neural networks [13].

9.4.1 Convolution for image processing

Image classification and object recognition are mainly applied to natural images. These types of images are *stationary*, which means that the statistics of one part of the image are the same as for another part of the same image. This allows us to use features extracted from one part of the image in other parts and apply feature detectors anywhere in the image. To train or create this small feature detector, we can extract small patches from the original images, and then convolve the features extracted with the larger image.

A *convolution* is done by multiplying the value of a pixel and its neighboring pixels by a matrix. In figure 26 we can see an example of how a 3×3 patch defined as is convolved to a 5×5 image.

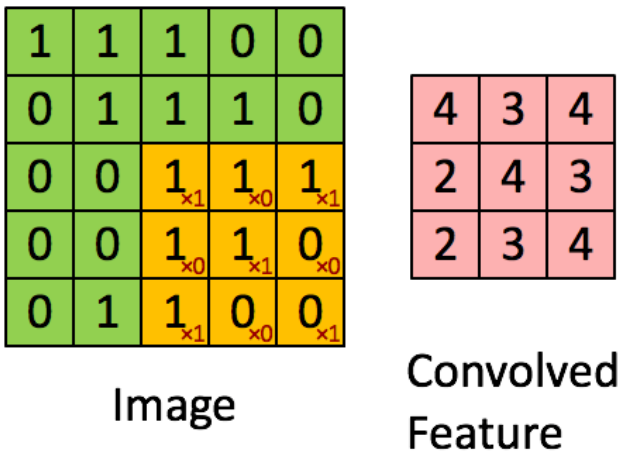

Figure 26: Example of a convolved feature.

9.4.2 Visual comparison of features extracted with ICA and RICA

For this part, we have sampled 200000 random 9×9 patches from the original MNIST dataset, and applied both ICA and RICA to them. What is shown in figures 27 and 28 are the elements of W extracted from both ICA and RICA when applied to these patches. What we see is a set of edge detectors that correspond to different edges that can be found in different parts of the original images. These representations would then be used with convolution methods to create a classification algorithm.

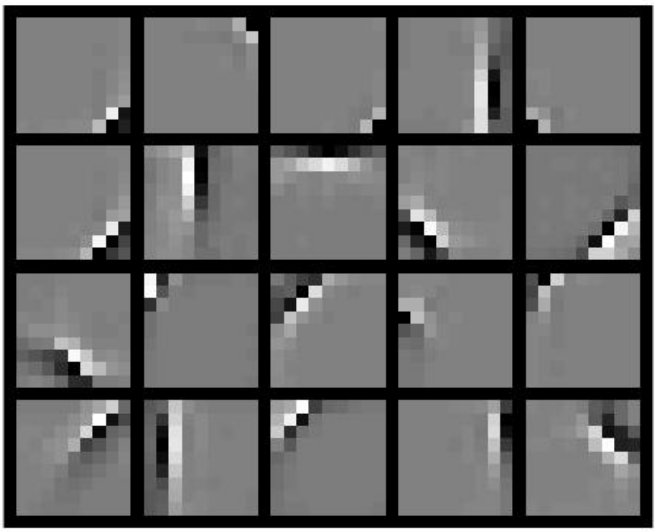

Figure 27: ICA extraction of 20 features.

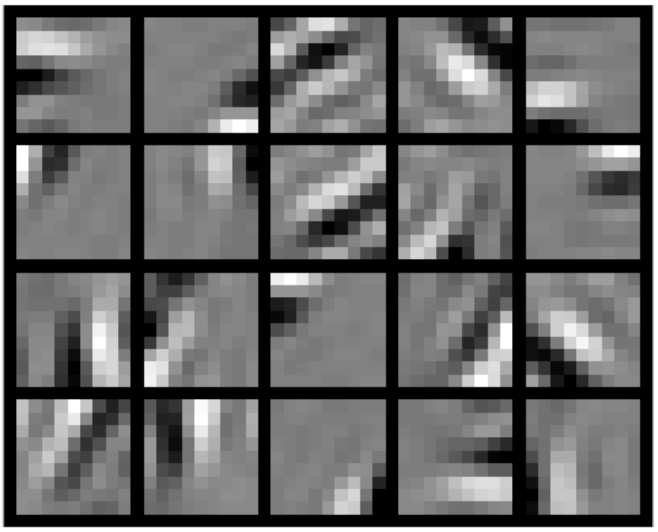

Figure 28: RICA extraction of 20 features.

The main difference between both visualizations is how ICA returns cleaner and sharper edges while RICA returns more blurred ones. This could be due to the faster convergence of the FastICA algorithm when compared to the RICA implementation.

9.4.3 PCA

Given that we have studied PCA and have used PCA whitening during this project, we can also take a look at the basis vectors that PCA extracts and see how they compare to those from ICA and RICA. In figure 29 we can see the first 20 principal components extracted from the same sample of patches as those used in the previous section.

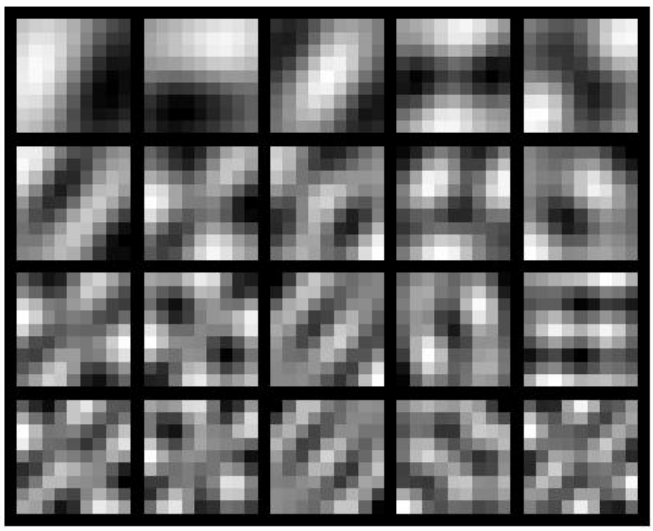

Figure 29: First 20 Principal Components extracted using PCA

The main difference between this basis and those extracted using ICA and RICA is that in this case we can see how the first vectors of the basis give the most information about the patches, while the rest give less information. Furthermore, using PCA we do not extract independent components as is the case in ICA and RICA.

10 Conclusions

10.1 Proposed objectives

The main objective was to review all the different algorithms and methodologies that were needed in order to understand Reconstruction ICA, and to create a functional implementation so that it could be used for feature extraction tasks. Once implemented, this allowed us to study its behavior when varying parameters of the implementation and to compare the reconstruction with the classical machine learning methods studied during the project. We were able to do this by using a simple database of handwritten digits. The fact that it was comprised of simple and relatively small images allowed us to perform many tests in order to evaluate how the algorithm worked.

10.2 Future work

Future work regarding this project comes from a practical application of the studied algorithm. We have tested the implementation and studied its characteristics, but have not applied it to what would be a more complex problem. The following steps would be to access a database of complex images such as fMRi scans and analyzing the results extracted by Reconstruction ICA, and compare them if possible to those extracted from ICA. Furthermore, we could study if properties such as overcomplete basis help with classification problems in fMRi, which would show an advantage over using conventional ICA.

References

[1] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen. Unsupervised Feature Learning and Deep Learning Tutorial. http://ufldl.stanford.edu/tutorial/ *Accessed June 2017*

[2] Jarrett, K., Kavukcuoglu K., Ranzato, M., i LeCun, Y. What is the best multi-stage architecture for object recognition? *IEEE Proc. International Conference on Computer Vision.* 2009. pp. 2146-2153

[3] Basheera, I. A. i Hajmeerb, M.. Artificial neural networks: fundamentals, com-puting, design, and application. *Journal of Microbiological Methods.* 2000, Vol.43, pp. 3-31

[4] Jan A. Snyman. Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms. *Springer Publishing.* ISBN 0-387-24348-8

[5] Qiao, Yu (2007). "THE MNIST DATABASE of handwritten digits". Retrieved May 2017.

[6] Apaar Sadhwani, Apoorv Gupta. Nonlinear Extensions of Reconstruction ICA. 2011

[7] Vćlav Hlavć. Principal Component Analysis - Application to images *Czech Technical University in Prague*

[8] Karhunen J, Oja E, Wang L, Vigario R, Joutsensalo J. A class of neural networks for independent component analysis *IEEE Transactions on Neural Networks* May 1997, 486 - 504.

[9] A. Hyvarinen, E. Oja. Independent Component Analysis: Algorithms and Applications. *Neural Networks*, 2000, 411-430.

[10] A. Coates, H. Lee, and A. Y. Ng. An analysis of single-layer networks in unsupervised feature learning. *AISTATS 14*, 2011.

[11] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[12] Quoc V Le, Alexandre Karpenko, Jiquan Ngiam, and Andrew Y Ng. ICA with reconstruction cost for efficient overcomplete feature learning. *Advances in Neural Information Processing Systems*, 24:1017 - 1025, 2011.

[13] Mattis Paulin, Matthijs Douze, Zaid Harchaoui, Julien Mairal, Florent Perronnin, et al.. Local Convolutional Features with Unsupervised Training for Image Retrieval. ICCV 2015 - IEEE *International Conference on Computer Vision*, Dec 2015, Santiago, Chile. IEEE, pp.91-99

[14] M. Schmidt. minFunc: unconstrained differentiable multivariate optimization in Matlab. http://www.cs.ubc.ca/ schmidtm/Software/minFunc.html, 2005. *Accessed June 2017*

[15] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen. Unsupervised Feature Learning and Deep Learning Tutorial. http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial *Accessed June 2017*

[16] Calhoun VD, Liu J, Adali T. A review of group ICA for fMRI data and ICA for joint inference of imaging, genetic, and ERP data. NeuroImage. 2009;45(1 Suppl):S163-S172.